

## Future Work

The library is currently tested on HPUX, Linux, NetBSD and Solaris operating systems. Future work includes making these and other operating systems effort-free compile targets. Also, the interface should be both finalized but extensible for future improvements. Another feature that should be added is the ability to leave a trail of pixels to show the path that the stroke has taken. Finally, other reference applications are being considered including “PCB”, a printed circuit board CAD program.

## Acknowledgments

The author would like to thank his friends and colleagues for their suggestions and criticisms during the creation and release of the LibStroke library. In addition, the programs that currently exist that implement similar algorithms were an inspiration to the author in creating this freely-available library. Finally, thanks goes to Purdue University, the Electrical and Computer Engineering Department, and the Student Organizations in the ECE department for creating an atmosphere in which creativity and motivation may produce innovation.

## References

The LibStroke library and reference application may be obtained from the LibStroke home page at <http://etla.ml.org/libstroke>.

## Results

The efficiency of the algorithm is  $O(n)$ , where  $n$  is the number of points in the stroke. The accuracy of the algorithm has also proven to be quite good. A reference application, FVWM2-Beta (a window manager) has been modified to use the Stroke Interface and the usability of the interface is exceptional. The true test of the usefulness of the interface is this anecdotal evidence: the author feels confined and impeded when using a non-strokes configuration! The final result to report is that the changes to the reference application are minimal. The patch kit to modify the original FVWM2 is approximately 500 lines long, including duplicate information used by the patch program and a good piece of configuration file as well. Most of the code changes were duplications of procedures used to parse the configuration file and bind actions to the stroke commands. An example of the sorts of actions that the reference application is capable of performing are shown by this excerpt from the FVWM2 configuration file:

### # Strokes

#	num	button	context	mod.	action
Stroke	14789	2	A	N	Exec exec xlock -mode blank&
Stroke	258	2	A	N	Exec exec xterm&
Stroke	563214789	2	A	N	Exec exec exmh&
Stroke	7415963	2	A	N	Exec exec netscape&
Stroke	741236987	2	A	N	Destroy
Stroke	1478963	2	A	N	Popup "Apps"
Stroke	74123	2	A	N	Module "winlist" FvwmWinList
Stroke	74159	2	A	N	Move
Stroke	852	2	A	N	Menu "Apps" Nop

(Some duplicate strokes included for accuracy reasons have been omitted for clarity.) The "num" field of the configuration file describes the path that the stroke takes through the matrix and the "action" field corresponds to the window manager function that is bound to that stroke. Virtually any window manager function may be bound to a stroke.

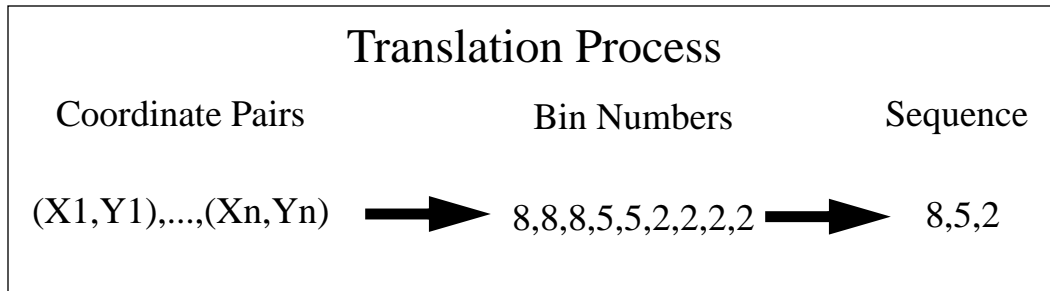


Figure 2: Steps in the Translation Process

## Implementation of the LibStroke Software Library

The LibStroke library consists of the following functions:

`stroke_init()` -- initialize the library and begin accepting stroke data

`stroke_record()` -- record a coordinate pair in the stroke path

`stroke_trans()` -- translate the given coordinates into a sequence

The `stroke_init()` function requires no special explanation. Its function is to initialize data structures and it does nothing remarkable. The `stroke_record()` function implements a simple linked-list append function. For efficiency, the `stroke_record()` function also maintains some statistics about the stroke in progress, such as minimum and maximum coordinate values and the running total number of coordinate pairs in the stroke. Because the library may receive coordinates sampled at a low rate (i.e. the mouse is moving faster than the X11 Window System is sending pointer motion events to the application), the `stroke_record()` function also generates interpolated points to fill in those gaps.

After the stroke is completed, the `stroke_trans()` function is called and the coordinate pairs are translated into a numeric sequence that describes the path of the stroke. The calling program then acts upon that command accordingly. Because of the bookkeeping done in the `stroke_record()` function, this can be done in one traversal of the linked list of coordinate pairs. Freeing the memory used by the coordinates is also performed in this pass.

1	2	3
4	5	6
7	8	9

Figure 1: Matrix of Position Bins

The upper and lower bounds of the matrix are defined by the maximum and minimum vertical coordinates in the stroke sequence. The left and right bounds of the matrix are likewise determined. The special case of an extremely tall or an extremely wide stroke needs to be given further treatment. If such an unbalanced stroke is encountered, it is most likely an attempt at drawing a straight line that has some error in the perpendicular dimension caused by imperfect control of the pointer by the user. Therefore, if the maximum horizontal delta between any two points in the stroke is more than four times the maximum vertical delta of any other two points in the stroke, (the stroke is wide and short), then the dimension of the horizontal side of the matrix is also used for the vertical dimension. This operation is likewise considered for the case when the stroke is tall and narrow.

Once the matrix has been defined, each coordinate pair is assigned a bin number according to its position in the matrix. This sequence of bin locations is then processed with a low-pass filter to remove any short excursions into other bins which have an insignificant number of coordinate pairs falling into them. Then, multiple adjoining occurrences of bin numbers are compressed into single digits. In this way, the sequence of bins that the stroke has passed through is determined. Figure 2 shows the flow of data from beginning to end.

## The Stroke Interface

The Stroke Interface is simple. To issue a command, the user presses a button on the mouse and drags the pointer through a path on the screen that corresponds to a command. The computer recognizes the pattern that the user has entered and responds appropriately. This sort of interface is often used in CAD tools. In a CAD tool, the user performs actions on selected objects. Deletion, instantiating, copying, and moving of objects are some common examples of those actions. When the user is designing with a CAD tool, cumbersome menus and awkward key sequences get in the way of the creative process. The Stroke Interface provides a much more natural and efficient command issue mechanism. If the user wants to delete an item, that item is selected with the pointer and then the user may “draw” a letter “D” (for “Delete”) with the pointer. Similarly, actions such as copy and unselect may be bound to a stroke that resembles the first letter of the word for the action. Other actions such as “draw a wire” or “cancel command” may be bound to simple strokes that may be easily issued. The ease of use of this interface has prompted the author to develop an efficient stroke recognition algorithm and implement that algorithm in a freely-available software library.

## The Recognition Algorithm

The algorithm described in this section is presented in a conceptually simple progression. The efficiency is improved in the implementation section of this paper. The algorithm is presented in a manner not directly amenable to efficient implementation here for the sake of clarity and ease of understanding.

The recognition algorithm processes a stream of coordinate pairs that describe the ordered path that the pointing device followed when the stroke was entered. These coordinate pairs are assigned to “bins” -- regions of the stroke defined by a three by three matrix superimposed over the region of space in which the stroke travels. Figure 1 is a representation of this matrix.

# Design and Implementation of a Stroke Interface Library

Mark Willey  
Department of Electrical and Computer Engineering  
Purdue University  
willey@purdue.edu

## Abstract

This paper describes an algorithm for translating the path of a pointing device into a numeric sequence that may be interpreted by a computer as a command. The implementation of this algorithm as an interface extension library for the X11 Window System is also detailed.

## Introduction

Human-computer interfaces provide mechanisms for an operator to interact with the machine. In the early years of computing, this interface was no more than the physical wiring of the computer's circuitry. Stored-program computers required a more sophisticated interface -- punched cards and teletype terminals were developed to allow the operator to provide more flexible inputs to the machine. Today, the most common input device is still a teletype keyboard. This keyboard is commonly supplemented with a pointing device such as a mouse. However, most system software and application programs ignore a significant aspect of motion pointer devices. The typical use of a pointer is simply to point at and select objects. This is known as the point-and-click model. This interface is, in essence, a teletype interface with a dynamic keyboard layout. The "keys" on the screen change position and function, but the user is still simply "pressing buttons". The interface described in this paper uses the motion pointer in a far more powerful way. The path of the pointer is translated into a command that the machine recognizes and acts upon. The "Stroke Interface" is a form of "gesture" recognition -- a simplification of handwriting and full gesture interpretation.